云环境中基于相对索引散列树的数据审核方法

李孟庭，周安宁

(广东外语外贸大学，广州 510006)

摘 要：为了确保云环境外包数据不受窜改，提高数据完整性审核的效率，提出一种基于相对索引散列树(RI-MHT)的数据审核方法，首先修改经典 MHT 的每个节点以存储两条信息，即数据块的哈希值和节点的相对索引，将 MHT 与节点的相对索引集成，以降低数据块搜索的计算成本；然后通过添加数据的最后修改时间，确保数据的新鲜性。实验结果验证了所提方法的有效性，与其他同类方法相比，所提方法在计算成本、通信成本和存储成本方面具有一定优势，并以较高的概率检测服务器的不当操作。

关键词：云环境；数据完整性审核；相对索引；散列树；计算成本

中图分类号：TP391 doi: 10.3969/j.issn.1001-3695.2017.11.1002

Data auditing method based on relative index hash tree in cloud environment

Li Mengting, Zhou Anning

(*Guangdong University of Foreign Studies Guangdong* 510006, *China*)

Abstract: To ensure the cloud environment outsourcing data from tampering, and improve the efficiency of data integrity audit, this paper proposed a data audit method based on relative index-merkle hash tree (RIMHT) . Firstly, it modified each node of the classic MHT to store two information, that was data block hash value and the relative index value of node. To reduce the computation cost of data block search, it Integrated the relative index of MHT with the node. Then, by adding the last modification time of the data, it ensured the freshness of the data. The experimental results verify the effectiveness of the proposed method. Compared with other similar methods, the proposed method has some advantages in terms of computational cost, communication cost and storage cost. And it is possible to detect the improper operation of the server with higher probability.

Key Words: cloud environment; data integrity audit; relative index; hash tree; computational cost

0 引言

云计算[1]在市场中迅速兴起，极大地改变了服务供应商与客户、经销商、雇员之间的交互方式，如百度、阿里巴巴等各种互联网服务已经改变了人们的交流、购物、教育和其他很多活动方式。云端提供的服务[2] 包括软件即服务（SaaS）、平台即服务（PaaS）以及设施即服务（TaaS）。云作为一个基础设施提供了远程存储位置以保存数据，并由云自身来维护。虽然云提供了存储便利，但数据安全性是其一个薄弱环节。

针对数据存储安全，已经有一些研究者对其进行了分析和开发。例如文献[3]提出了两个数据持有型的验证方案，分别为采样和高效方案。但数据所有权的保证较弱，使用了基于 RSA[4]的同态标签。然而该协议不支持数据的动态操作，其应用仅限于静态数据。文献[5]提出了两个可检索证据（PoR）方案，使用了支持公共审核的同态认证器。该研究基于可公共审核的 BLS 签名。该方法在审核程序的时间复杂度方面具有高效性，然而该协议的通信复杂度会随着外包数据块大小的增加而呈线性变化。文献[6]提出了动态数据的处理程序机制，该方法依赖于基于排名的身份验证的跳跃表，虽然跳跃表改进了标签生成方法，但其效率没有验证。文献[7]基于 Merkle 散列树和附属设计模块，确保设备主动、周期性地向管理中心发送自己的平台运行状态,使得存储容量开销和计算成本所需减少，从而减轻了管理中心的计算压力，但该研究仅限于静态数据。文献[8]基于服务器持有数据证据的正确性和完整性，提出一个改进版的协议。虽然该协议在加密外包数据库中具有较好的安全性，但其应用仅限于静态数据。

现有的协议试图为云系统提供数据完整性审核，但尚未能解决公共审核和数据动态性的问题。为此，本文提出了一种数据完整性审核协议，带有相对索引 Merkle 散列树（RIMHT），其支持了数据的公共审核和数据动态操作，并能够确保数据的新鲜性，

作者简介：李孟庭（1974-），男，广东广州人，讲师，硕士，主要研究方向为多媒体技术、数据存储与分析等（iamprofli@163.com）；周安宁（1966-），男，广东广州人，副教授，博士，主要研究方向为大数据存储与分析等.

还考虑到了外包数据的隐私保护问题。

1　提出的数据完整性审核模型

提出的数据完整性审核模型如图 1 所示。图 1 中的数字表示在一次数据审核过程中的作业顺序，“*”表示该活动可以独立于审核过程之外。该审核模型包含三方实体，其定义如下：

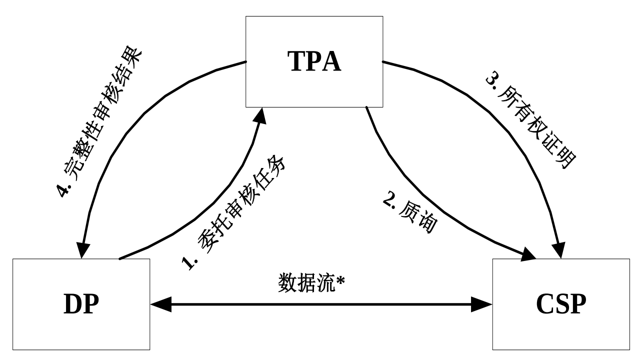


图 1　RI-MHT 的系统模型

a）数据所有者（DP）。通过修改、插入、附加等操作在后续对数据进行更新。DP 是依赖云供应商提供的数据维护实体。通常来说，是一个资源受限的实体。

b）云服务供应商（CSP）。拥有足够计算资源和无限存储空间的存储服务器实体。主要负责外包数据的保存和维护，但 CSP 一般被考虑为一个不可信的实体。

c）第三方审核者（TPA）。在 DP 的数据上进行审核的专业实体。TPA 是 CSP 和 DP 都信任的实体，能够最大限度降低 DP 的数据审核计算压力。

TPA 是可信实体，CSP 为不可信实体。然而 TPA 和 CSP 都可能给 DP 数据带来一些威胁，例如：

a）TPA 造成的数据安全威胁。DP 依靠 TPA 来确保数据的完整性，并假定 TPA 是一个真实可靠的独立实体。然而 TPA 始终存在着窥探 DP 数据的可能性，因此，在公共审核协议中，数据的隐私性可能会受到 TPA 的侵犯。本文提出的协议未考虑数据隐私性问题，并假定 TPA 忠诚可靠。

b）CSP 造成的安全威胁。CSP 可能会对 DP 的数据造成如下威胁： CSP 可能会在没有通知 DP 的情况下移除其不常访问的数据，以节约服务器空间；CSP 可能会对数据造成一些处理错误，这可能会使 DP 数据出现不可恢复的损坏。

c）一些外部威胁。合法用户可以通过 CSP 提供的应用程序访问外包数据，而较弱的应用程序可能为 DP 数据带来风险。一些合法用户的登录凭证可能会被陌生人使用，并可能匿名地污染或删除数据。另外，离职后的 CSP 管理员可能会入侵云服务器，对存储数据造成危害。因此，在提供各种云服务的同时，确保数据的可访问性并防止外包数据受到外部攻击，对于 CSP 是至关重要的。

2　提出的协议

本章将解释在云计算中执行数据完整性审核的 RI-MHT 协议。首先，对经典 MHT[9]进行修改；然后，详细介绍所提协议的构建。本文系统流程如图 2 所示。该流程基本遵循图 1 模型，将数据流进行了展开。

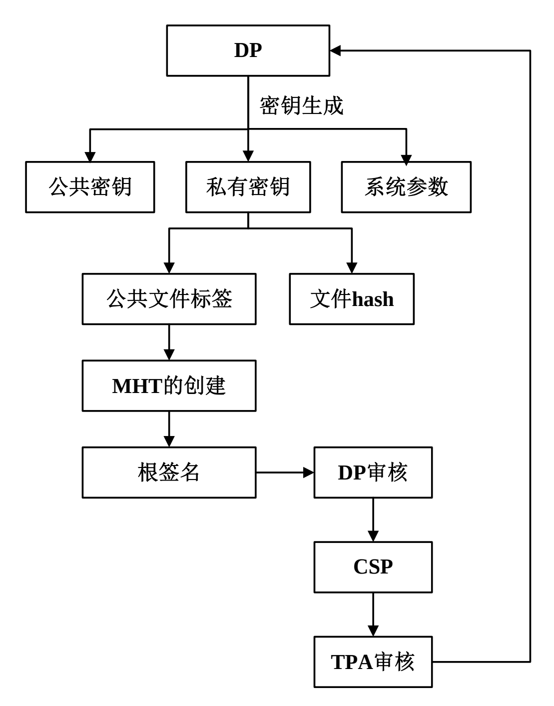


图 2 本文系统基本流程框架

2.1 MHT 的修改

为降低 MHT 中对数据进行完整性审核时搜索节点的计算复杂度，本文对经典 MHT 的每个节点进行修改以存储两条信息，一条信息为数据块的 hash 值，另一条信息为节点的相对索引。与一个节点 P 相关联的相对索引，指定了属于 P 子树的叶节点数量。在修改后的 MHT 中，叶节点的相对索引设为 1。举例来说，如果对于一个父节点 P，L 和 R 的 hash 值分别为 H_a 和 H_b、相对索引字段值为 r_a 和 r_b 的父节点 P 的左子节点和右子节点，那么节点 P 的 hash 值为 $(H_a \| H_b)$，相对索引字段值为 $(r_a + r_b)$。将一个时间戳字段与 MHT 的根节点相关联。修改后的 MHT 样例如图 3 所示。此处，H_R=$(H_a \| H_b \| d_t)$，式中 d_t 为树建立的日期和时间。由于树中任何数据块进行的改动均会更新 H_R，所以 H_R 中能够反映出最后修改的时间和日期，这样就保证了数据的新鲜性。

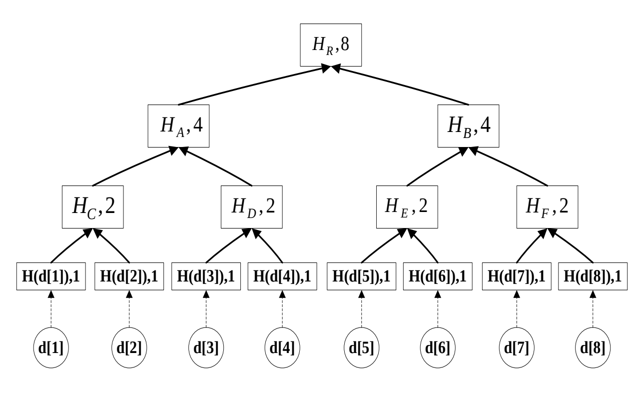


图 3 带相对索引和时间戳的 merkle 散列树

2.2 定义

本节将给出所提算法的的定义。

a）Keygen（1^{λ}）：该算法由 DP 执行。其中，λ 表示安全性参数，该算法的输出为一个密钥对（公钥，私钥）$\leftarrow(\eta,k)$。

b）FileTagGen(fname,k,n,d_t)：该算法由 DP 执行，以生成文件 F 的标签。该算法的输入为外包文件的名称、私有密钥、数据块分区数量、文件预处理的日期和时间。该文件标签表示为 τ 。

c）BlockSigGen$(k,H(d[i]),d_t,u)$：该算法由 DP 执行。其输入为私有密钥 k、文件块的 hash、文件预处理的日期和时间，以及一个随机元素 $u \in G$。该算法输出为 θ，是文件块的上 BLS 签名[10] $\{\psi\}_{1\le i\le n}$ 的一个有序集合。

d）Challenge（质询）：该算法由 TPA 执行，以在 DP 委托审核权之后生成发送到 CSP 的质询消息。

e）GenProof(F,θ,C)：CSP 在接收到来自 TPA 的质询消息 C 后立即执行该算法，以生成一个证据 P_f，并将其传递到 TPA 以进行验证。该算法的输入为文件 F、签名集合 θ 和质询消息 C。

f）VerifyProof(C,P_f,η)：该算法由 TPA 执行，以验证从 CSP 接收到的证据 P_f。该算法的输入为质询消息 C、来自 CSP 的消

息 P_f 以及公钥 g^k。该算法的输出为{TRUE,FALSE}，其取决于验证的成功或失败。

2.3 构建细则

F 表示要外包的文件，该文件被分为 n 个数据块 $(d[1],d[2],......,d[n])$。假定 $e:G\times G\rightarrow G_T$ 表示一个双线性映射，其中 g 为生成器，p 为群 G 的素数阶。设 $H:\{0,1\}^*\rightarrow G$ 为加密 hash 函数。本文所提协议如下：

a)Keygn 1^λ ）。首先，DP 必须使用算法 1 展示的密钥生成程序，生成密钥集 (pubkey, seckey)。该算法首先选择一个随机元素 $k\in Z_p$ 作为一个私有密钥，并生成 $\eta=g^k$ 作为公钥。

算法 1 生成算法

1. procedure KEYGEN (1^λ)

Input：安全性参数 λ

Output：发布系统参数 (G,g,η)，保留 k。

2. 初始化配对：
3. pbc_demo_pairing_init(pairing, count, param)
4. 初始化 G 和 Z_P 的元素
5. element_init_G(g, pairing)；
6. element_init_G(η , pairing)；
7. element_init_ Z_P (k , pairing)；
8. 生成系统参数和私有密钥
9. $g\leftarrow$ element_random(g)；
10. $k\leftarrow$ element_random(k)；
11. 生成公共密钥
12. η $\leftarrow$ element_pow_ $Z_P(\eta,g,k)$.

b)FleTagGen。由于 DP 要生成文件 F 的标签，所以 DP 首先生成一个随机元素，$u\in G$。DP 生成系统日期和时间，表示为 d_t。将系统日期和时间串联在文件标签 τ 中，以确保数据文件的新鲜性。设 $h=(\text{fname}\,\|\,n\,\|\,u\,\|\,d_t)$，其中 $h\in G$，$\tau=sig_k(h)$ 为 F 的文件标签。对串联的字符串 h 进行本地存储，以用于未来的文件标签验证。生成文件标签的伪代码如算法 2 所示。

算法 2 生成文件标签

procedure GENERATE_FILE_TAG

Input：文件名 fname，私钥 k，文件块数量 n，日期和时间 d_t

Output：公布文件标签(τ)。保留文件标签 hash (h)。

2. 初始化配对：
3. pbc_demo_pairing_int(pairing, count, param)
4. 对 G 的随机元素 u 和 h 进行初始化：
5. element-init_G(u, pairing)；
6. element_init_G(h, pairing)；
7. 生成 u
8. g $\leftarrow$ element_random(u)
9. 读取系统日期和时间
10. d_t $\leftarrow$ ctime(t)
11. 串联 fname、n、u、d_t
12. file_tag_concatenation $\leftarrow$ fname ‖ n ‖ u ‖ d_t
13. file_tag_concatenation 转换为随机元素 $h\in G$；
14. $h\leftarrow$ element_from_hash (file_tag_concatenation)；
15. 生成文件标签 τ

```
τ ←element_pow_ Z_P ( τ ,file_tag_concatenation, k)
```

c)BlockSigGen。在生成文件标签 τ 后，DP 为每第 i 个文件块 $d[i]$ 生成 BLS 签名 $\psi_i=(H(d[i]).u^{d[i]})^k$ ，其中 $i\in\{1,2,\ldots,n\}$ ，$u\in G$ 。此处 $H(d[i])$ 表示使用一些加密 hash 函数得到的文件块 hash 值。提出的协议中使用了 SHA256 hash 算法。签名集合表示为 $\theta=\{\psi_i\}$ ，$1\le i\le n$ 。在生成块签名后，DP 生成一棵 RI-MHT，使用 $H(d[i])\forall i\in\{1,n\}$ 作为叶节点，并生成一个根节点 H_R 。在实现 H_R 时，DP 将其与系统日期和时间相串联，以确保数据新鲜性。由此，$H_R=H_R\parallel d_t$ 。现在 DP 使用 k 对根 H_R 进行验证。根签名最后表示为 $\rho\leftarrow H(R)^k$ 。DP 将向 CSP 发送 $\{F,\tau,\theta,\rho\}$ 信息以从本地存储中对相同的信息进行存储和删除。本文 MHT 创建算法的伪代码如算法 3 所示。

算法 3　MHT 创建算法

```
procedure MERKLE_HASH_TREE_CREATION
Input：结构元素 mt，块 hash (H(d[i]) ,
头标文件 merkle_tree_h
Output：根节点(roothash)
将 leaf_start 初始化为块数量
    leaf_start ← (1 ≪ (mt → treeheight − 1)) ;
初始化树中的节点数量
    mt → n ← (leaf_strat + mt → data_blocks－1)
 为 mt →节点中的 mt 分配内存
 初始化 leaf_start 计数
 当该计数小于或等于 mt → n
   设 mt →node[count].index  为 1；
    设 mt →node[count].hash 为 H(d[count－leaf_start])；
   设 mt →node[count].left  为 NULL；
   设 mt →node[count].right  为 NULL；
   计数以 1 为增量；
 end while
 将计数初始化为 leaf_start -1
 当计数大于 0 时
    设 mt →node[count].hash 为 NULL 值；
    设 mt →node[count].hash 为 mt →node[2*count].hash
+mt →node[2*count+1].hash；
   设 mt→node[count].index 为 mt→node[2*count].index
+mt →node[2*count+1].index；
   设 mt→node[count].left 为 mt→node[2*count].left；
    设 mt→node[count].right 为 mt→node[2*count].right；
   计数以 1 为减量；
   若计数等于 0；
   返回地址 mt → node
 end while
```

d）质询。DP 可能会对外包数据 F 的完整性进行检查。为此，DP 通过在一个安全可靠的信道上共享文件细则、根签名和私有密钥，将审核任务委托给 TPA。为开始审核，TPA 选择随机的包含 k 个元素的子集 $Q=\{r_1,r_2,\ldots,r_k\}\in[1,n]$ 和 $i\in Q$ 。TPA 将质询消息 $C\leftarrow(i,b_i)_{i\in Q}$ 发送到 CSP。

e）GenProof。一旦接收到来自 TPA 的质询消息 C 后，CSP 立即使用搜索算法对 hash 树中第 i 个节点 $\forall i\in C$ 进行搜索。在进行节点搜索时，CSP 可以在第 i 个节点的搜索路径上存储每个兄弟节点的信息。该信息将作为正被搜索节点的辅助信息（AI）。

举例来说，若第 i 个节点为图 3 中的节点 5，则其 AI 为：$[(H_A,4,L),(H(d[6],1,R)(H_F,2,R)]$，其中 L 表示左兄弟节点，R 表示右兄弟节点。若 MHT 中不存在第 i 个节点，则 CSP 向 TPA 通知块未发现的消息，并停止审核过程；若块确实存在，则 CSP 继续证据生成程序，并生成一个完整的所有权证据。

f）VerifyProof。在开始验证之前，TPA 先验证文件标签 τ 如下：

（a）恢复本地存储的文件标签 hash h。

（b）恢复来自服务器的文件标签 τ 。

（c）检查公式 $e(h,g^k)\overset{?}{=}e(\tau,g)$ 是否成立。

（d）若验证通过，则输出 TRUE，并恢复 u 和 d_t 。

（e）若未通过验证，则停止程序。

若文件标签 τ 的真实性验证未能通过，则 TPA 会停止进一步的审核程序，并将完整性审核过程失败的消息报告给 DP；若文件标签的认证通过，则 TPA 恢复 u 和 d_t 字段，以将其用于对接收自 CSP 的 P_f 中的根 ρ 进行认证。完整性认证的伪代码如算法 4 所示。

算法 4　完整性认证

1：　procedure INTEGRITY_VERIFICATION

Input：P_f，η，C

Output：　TRUE if 文件块通过认证　else FALSE

2：　初始化配对：

3：　　pbc_demo_pairing_init(pairing,count,param)

4：　初始化少数随机元素 G 的 temp1、temp2、temp3、$H(d[i])$、Power[i]，Z_P 的 b[i] ，其中 $i\in$ Q

5：　　element_init_G(temp1, pairing);

6：　　element_init_G(temp2, pairing);

7：　　element_init_G(temp3, pairing);

8：　　element_init_G($H(d[i])$, pairing);

9：　　element_init_G(Power[i], pairing);

10：　　element_init_Zr(b[i], pairing);

11：　　element_set1(temp1);

12：从服务器读取 H(d[i])、μ；从 C 读取 $b[i]$，u

生成 $Hd([i])^{b[i]}.u^{\mu\forall}$ ，$i\in$ Q

13：　　element_pow_Zp(Power[i], $H(d[i])$, $b[i]$);

14：　　element_pow_Zp(Power1[i], u, μ)

15：生成 $\prod_{i=r_1}^{r_k}(H(d[i])^{b_i}.u^{\mu}$

16：　　element_mul(Power[i], Power[i], Power1[i])

17：　　element_mul(temp1, temp1, Power[i]);

18：在 temp1 上应用配对

19：　　pairing_apply(temp2, temp1, η, pairing);

20：在 ψ 上应用配对

21：　　pairing_apply(temp3, ψ, g, pairing);

22：比较 temp2 和 temp3

23：　　result=element_cmp(temp2, temp3)

24：if 结果为 0 then output TRUE

25：else output FALSE

3　性能分析

本章将分析 RI-MHT 的性能。随着损坏块数量的不同，其计算复杂度的变化。还将给出不同算法的计算成本、存储成本和通信开销，以及与其他数据完整性审核协议之间的比较结果。

3.1　计算成本分析

审核模型中不同实体的计算成本是不同的。对于 TPA，计算成本指的是执行一次完整性审核所需的资源。对于 CSP，计算成本对应于处理质询消息，并生成所有权证据所需的时间。数据动态操作与常规数据完整性审核中产生的计算成本是不同的。在更新操作中，TPA 需要生成新鲜块的块标签，CSP 需要重新生成根 hash 并更新文件标签。所有这些任务均会对各自涉及的实体产生计算负担。

下面将给出 RI-MHT 的计算成本，并与文献[3]和文献[6]进行比较。为简便起见，分析实验使用表 1 给出的符号。RI-MHT 中每个算法的计算成本如表 2 所示。其中，n 表示数据块总数；t 表示质询数据块数量。上述三种协议的计算成本比较如表 3 所示。由表可知，所提协议的效率高于其他两个协议。文献[3]搜索数据块的计算时间较长，并根据块数量增长而线性变化，即 $O(n)$。所提协议中，由于使用了先对索引字段，搜索一个节点的复杂度被降低为 $O(\log n)$。所提搜索算法的计算复杂度与在包括$(2n\text{-}1)$个节点的二叉搜索树[11]（BST）中搜索一个元素相似，而文献[3]则与线性搜索算法的计算复杂度相似。图 4 给出了所提协议在频繁数据改动中的性能。实验中对 1 GB 的文件进行更新，其中更新的块数量从 100~1000 间变化。由图可知，随着被改动块数量的增加，文献[3]计算成本增加的速率高于本文协议，而文献[6]则不支持数据动态操作。

表 1　符号和描述

符号	描述
H(G)	将一个数值散列到群 G 中
Add(G)	群 G 中的加法任务
Mul(G)	群 G 中的乘法任务
Exp(G)	群 G 中的求幂任务
Pair(G)	群 G 中的配对任务

表 2　RI-MHT 中不同算法的计算成本

算法	加法	乘法	Hash	求幂	配对
KeyGen	0	0	0	Exp(G)	0
FileTagGen	0	0	0	Exp(G)	0
BlockSigGen	0	0	nH(G)	(n+1)Exp(G)	0
GenProof	tAdd(G)	(t-1)Mul(G)	0	tExp(G)	0
VerifyProof	0	tMul(G)	0	tEXP(G)	4

表 3　计算成本的比较

协议	服务器	审核者	时间复杂度
文献[3]	tEXP(G)+(t-1)Mul(G)	2Pair+(t+2)Exp(G)+(t+1)Mul(G)	$O(n)$
文献[6]	2tExp(G)+(2t-2)Mul(G)	3Pair+(t+1)Exp(G)+tMul	$O(n)$
RI-MHT	tExp(G)+(t-1)Mul(G)	4Pair+tExp(G)+tMul(G)	O(log$n$)

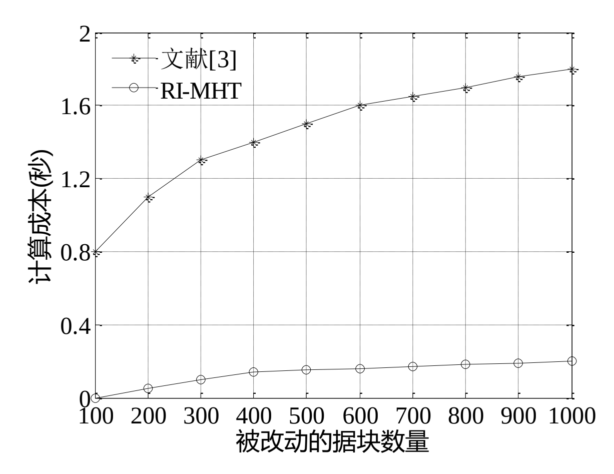


图 4 计算成本随被改动数据块数量的变化情况

3.2 存储成本分析

下面分析 RI-MHT 的存储和通信成本。DP 存储密钥对 (η,k)、文件标签散列 h。受委托的 TPA 存储公钥 η、文件标签 τ。CSP 存储 DP 的数据文件 F、文件块标签 θ、根签名 ρ、文件标签 τ。因此，在存储开销中，文件预处理时生成的元数据大小会产生重要影响。在基于 RSA 的协议中生成的元数据为 RSA 模的大小。考虑 RSA 模的大小为 2 048 位，公钥大小为 6 168 位素数，那么协议得出一个数据块的大小应该保持 $\leq e/2$。若一个数据块 $< e/2$，则协议可能会变得不安全。因此，一个 64 MB 的数据需要被分割为最小 87 056 个数据块，每个数据块的元数据长度为 2 048 位。由此，总元数据大小约为 21.2 MB。在 RI-MHT 中，本文选择基于 BLS 签名的同态标签，因为与基于 RSA 标签的同态标签相比长度较短。RI-MHT 中的存储成本如表 4 所示。其中，n 表示数据块总数量；k 表示安全性参数；l 表示文件标签的大小；|G|表示属于 G 的一个元素的大小。从表 4 中可知，在 DP 和 TPA 处由于元数据而产生的存储开销是恒定的，但在 CSP 处则呈线性变化。从表 4 还可知道，各协议的 TPA 存储成本是相同的，最大的区别在于数据所有者 DP，以及云供应商 CSP。因此，本文 RI-MHT 在总体存储成本上具有一定优势，这主要得益于生成的元数据较小。

表 4 RI-MHT 中的存储成本/bit

实体	本文	文献[3]	文献[6]
DP	k+2|G|+l	k+3|G|+2l	k+3|G|+l
TPA	k+|G|+l	k+|G|+l	k+|G|+l
CSP	n+l+(n+1)|G|	n+2l+(n+1)|G|	n+2l+(n+1)|G|

3.3 通信成本分析

在一个审核实例中 CSP 与 TPA 之间的数据传输量构成了通信成本。在 RIT-MHT 中，通信成本主要来自于 TPA 向 CSP 发送质询消息，以及 CSP 向 TPA 发送证据消息。DP 将审核任务委托给 TPA 所产生的通信开销是恒定的。RIT-MHT 中各方实体之间的通信开销如表 5 所示。其中，$|Z_p|$表示属于 Z_p 一个元素的大小；t 表示质询数据块的数量；l 表示文件标签的大小；|G|表示属于 G 的元素大小。

表 5 RI-MHT 中的计算成本

阶段	计算成本（bits）
委托审核任务	DP→TPA：$2\|G\|+l$
质询阶段	TPA→CSP：$t.\|Z_p\|+283$
响应证据	CSP→TPA：$\|G\|+\|Z_p\|+(t+2)256$

由于 DP 将审核任务委托给 TPA，DP 向 TPA 共享密钥、文件标签和根签名，所以该通信成本每次审核仅产生一次，等于密钥、根签名和文件签名的大小。TPA 向 CSP 发送包含 k 个元素的子集 Q、随机元素 $b_i \in Z_p, i \in Q$ 质询消息。相关研究表明[12, 13]，

Q 可表示为 $\{\log|F|+3\log(1/err)\}$ 字节，其中，$\|F\|$ 指的是数据文件大小；err 为误差概率。一般来说，对于一个文件大小 ≤ 1024 TB 和 $2\le err\le 80$，Q 的大小可表示为 283 Byte。因此质询阶段中，对于大小 ≤ 1024 TB 的一个文件，通信成本的最大值为 $t.|Z_p|+283$ Byte。

3.4　综合评价

本文进行实验的系统为 2.20 GHz Intel core(TM)2 双核处理器，2 GB RAM。使用 PBC 库[14]版本 0.5.14，散列操作使用配对和加密库 Openssl 版本 1.0.2e。协议构建中实施了 160 位群阶椭圆曲线。仿真平台为 ubuntu-10.04 操作系统。为进行实验演示，本文使用的文件大小为 10~100 KB。不同数量的文件块进行完整性审核时 TPA 的计算成本如图 4 所示。由图可知，使用 RI-MHT 时 TPA 的计算负担最小。为研究更新操作的影响，文献[3]协议中，TPA 首先搜索 MHT 中文件块的位置，其复杂度随树中块数量的增加而线性增长。由此，搜索操作为 TPA 带来了额外的计算开销。而在 RI-MHT 中，搜索一个节点的复杂度被对数降低了。从图 5 可知，文献[3]和文献[6]的计算成本随着文件大小的增大而升高，而本文方法对 TPA 造成的计算开销最小。因此，当 DP 将一个大数据文件外包到云中，或频繁更新云中的外包文件时，所提 RI-MHT 能够大幅降低计算开销。

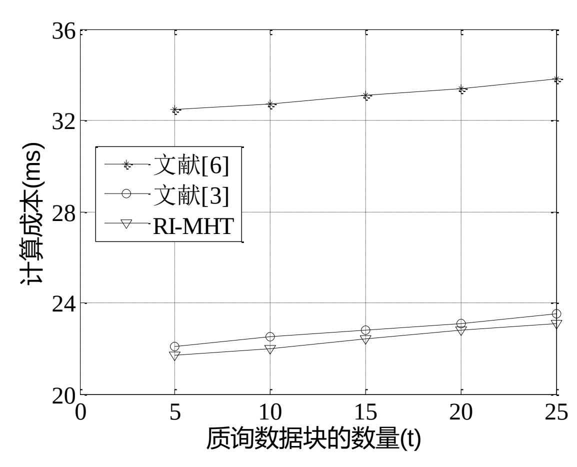


图 5　计算成本与质询数据块数量的关系

4　结束语

本文基于 MHT 构建和 BLS 签名，提出了一个云数据的完整性审核协议。为了降低搜索复杂度，节点的相对索引集成了散列树中的 hash 值。该协议将时间戳字段串联到根 hash 中，以确保数据的新鲜性，使得用户在任何时间访问的数据文件均是数据的最近一次副本。因此，所提协议支持高效数据公共审核，确保了数据的新鲜性。实验结果和安全性证明表明 RI-MHT 具有较高的安全性和效率。

参考文献：

[1] 张建勋，古志民，郑超．云计算研究进展综述 [J]．计算机应用研究, 2010, 27 (2): 429-433.

[2] Mell P, Grance T. The NIST definition of cloud computing [J]. Communications of the ACM, 2009, 53 (6): 50-50.

[3] Yao C, Xu L, Huang X, et al. A secure remote data integrity checking cloud storage system from threshold encryption [J]. Journal of Ambient Intelligence & Humanized Computing, 2014, 5 (6): 857-865.

[4] 许柯，刘绪崇，符振艾，等．网络信息加密 RSA 算法的运算速度和保密性优化 [J]．科技通报, 2015, 34 (7): 144-147.

[5] Shacham H, Waters B. Compact proofs of retrievability [J]. Journal of Cryptology, 2013, 26 (3): 442-483.

[6] Erway C C, Papamanthou C, Tamassia R. Dynamic provable data possession [J]. ACM Trans on Information & System Security, 2015, 17 (4): 15-26.

[7] 谢飞．基于 Merkle 散列树的可信云计算信息安全证明方法 [J]．激光杂志, 2016, 37 (11): 122-127.

[8] Wang J, Chen X, Huang X, et al. Verifiable auditing for outsourced database in cloud computing [J]. IEEE Trans on Computers, 2015, 64 (11): 3293-3303.

[9] 李凌．云计算服务中数据安全的若干问题研究 [D]．合肥：中国科学技术大学, 2013.

[10] 巩俊卿，钱海峰．具有完全保密性的高效可净化数字签名方案 [J]．计算机应用研究, 2011, 28 (1): 312-317.

[11] 王秋芬，梁道雷．一种构建最优二叉查找树的贪心算法 [J]．计算机应用与软件, 2013, 30 (7): 57-61.

[12] Etemad M M. Generic efficient dynamic proofs of retrievability [C]// Proc of ACM on Cloud Computing Security Workshop. London, UK: IEEE press, 2016: 85-

96.

[13] 刘爱分. 云环境下高效动态的密文搜索方法 [D]. 沈阳: 东北大学, 2013.

[14] Shelat A, Shen C H. Two-output secure computation with malicious adversaries [M]// Advances in Cryptology – EUROCRYPT 2011. Springer Berlin Heidelberg, 2011: 386-405.